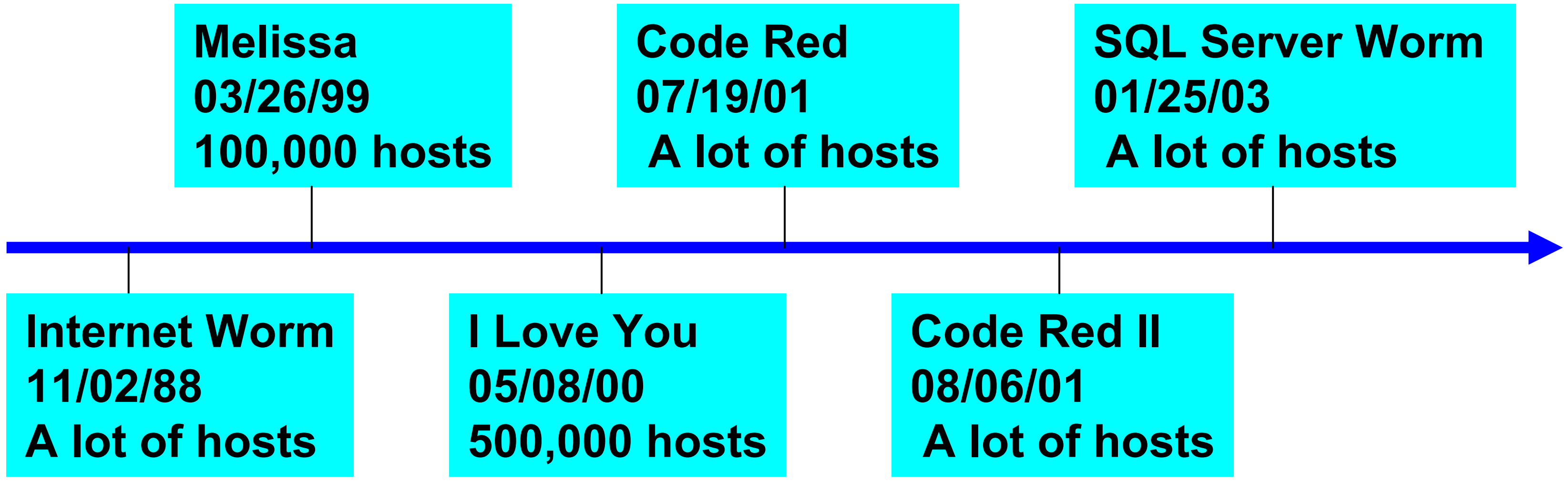
Defending Military Enterprise Networks Against Worm-based Attacks – The Self Healing Aspect

Peng Liu

pliu@ist.psu.edu

Pennsylvania State University

Intro (1)



Effects:

- Cause denial-of-service conditions
- Corrupt files
- Install Trojan Horses
- Can do almost everything bad on a host
- Disable a route to forward packets
- …

Intro (2)

Why are worms so difficult to prevent?

- There are (always) inevitable, unknown security vulnerabilities

As a result, you can prevent a worm from happening again, but you cannot prevent new worms!

- Worms are self-propagating (usually in a random way)
- Propagating is usually much quicker than detection and recovery

As a result, it could be too late when you detect a worm and take reactive defense actions!

Proactive defense is essential!

Motivation

During a war:

- A Military Enterprise Network (MilEN) delivers critical services
  - o command & control; intelligence analysis; logistics planning; etc.
- The goal of the opponent's worm can be
  - o disable the MilEN to deliver services – availability issue
    - ✓ cause denial-of-service
  - o mislead the MilEN to deliver wrong services -- integrity issue
    - ✓ data & code corruption; Trojan horses; etc.
- When you shut down the MilEN 6 hours to fix the worm
  - o Although you ensure that the MilEN will not deliver wrong services after it resumes, the opponent's real goal can be the 6 hour outrage

MilEN need not only service integrity, but also availability in the face of worm-based attacks!

Traditional worm recovery

1. "Something is wrong!"
2. Suffer; panic
3. Disconnect usually the whole subnet; disable a lot of local services, if not all
4. Analysis – focus on integrity issues
5. Repair
6. Fix the hole: reconfigure firewalls, install patches, …
7. Reopen the Internet connection

$\left.\begin{array}{l}3\text{–}7\end{array}\right\}$ **Recovery Time Window**

Too much availability can be lost during the recovery time window (24 to 48 hours for Penn State SQL Server Worm Recovery)!

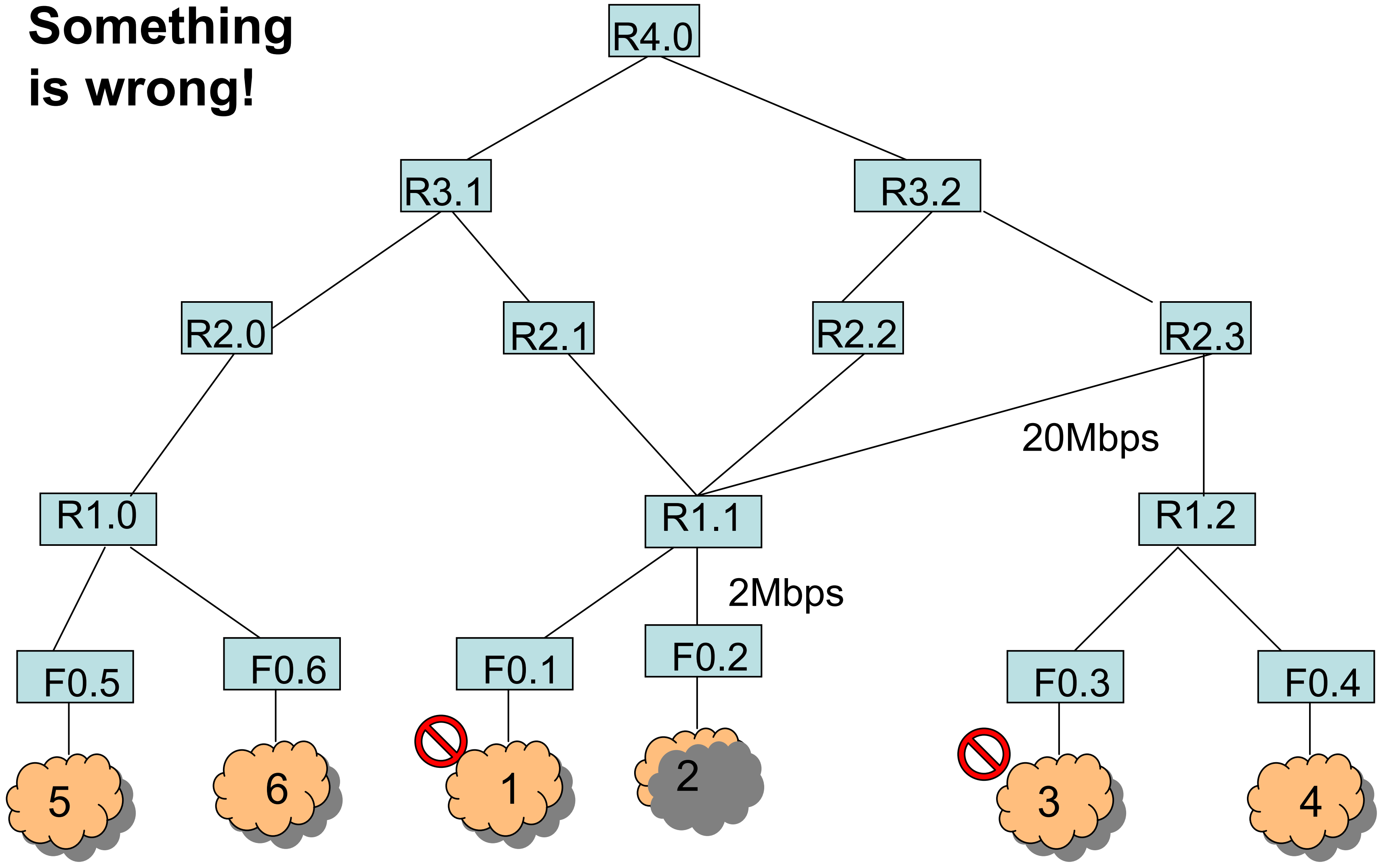
The goal of our approach

Traditional recovery	Our approach
Offline recovery	On-the-fly recovery or self-healing
Fix the hole offline	Enhance security on-the-fly
Will not reopen a connection unless the subnet is repaired & fixed	Reopen a connection as soon as the comprised part of the subnet is contained
Will not allow a system to deliver any service unless the system is repaired & fixed	Allow a system to deliver services as soon as the comprised part of the system is contained

Constraints:
-- We want availability, but we will not tolerate serious integrity loss

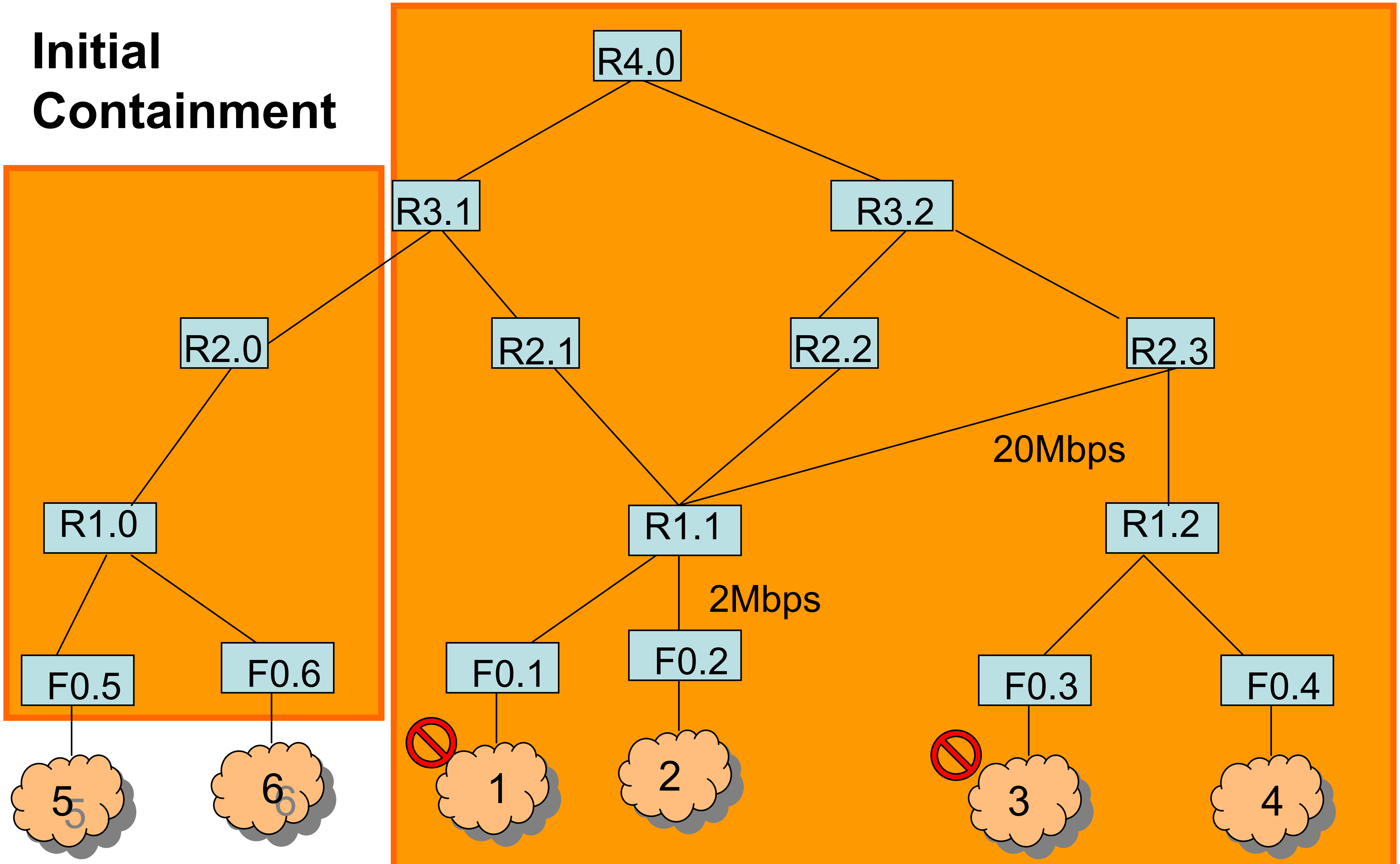
Our approach in a nutshell (1)

Something is wrong!



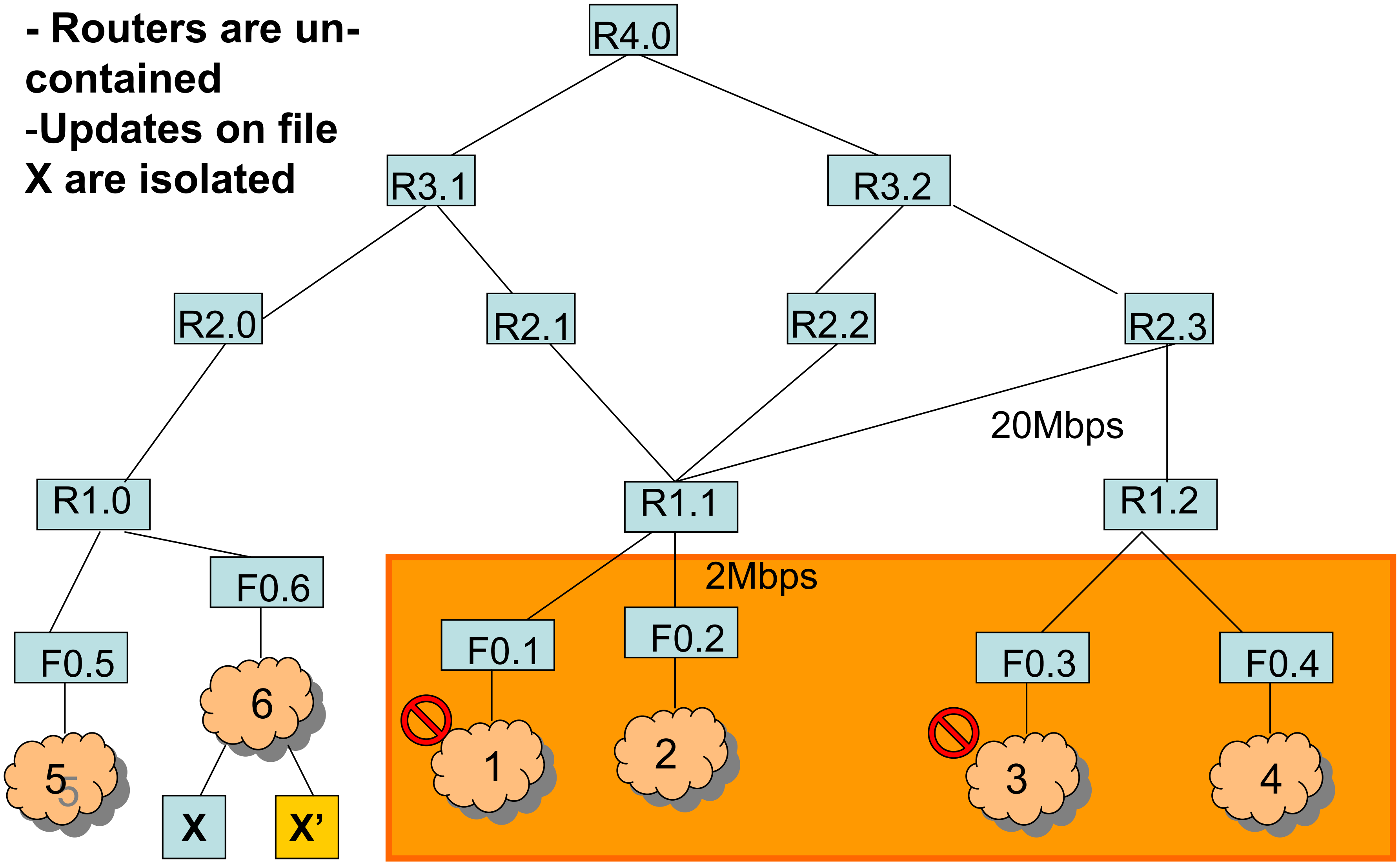
Our approach in a nutshell (2)

Initial Containment



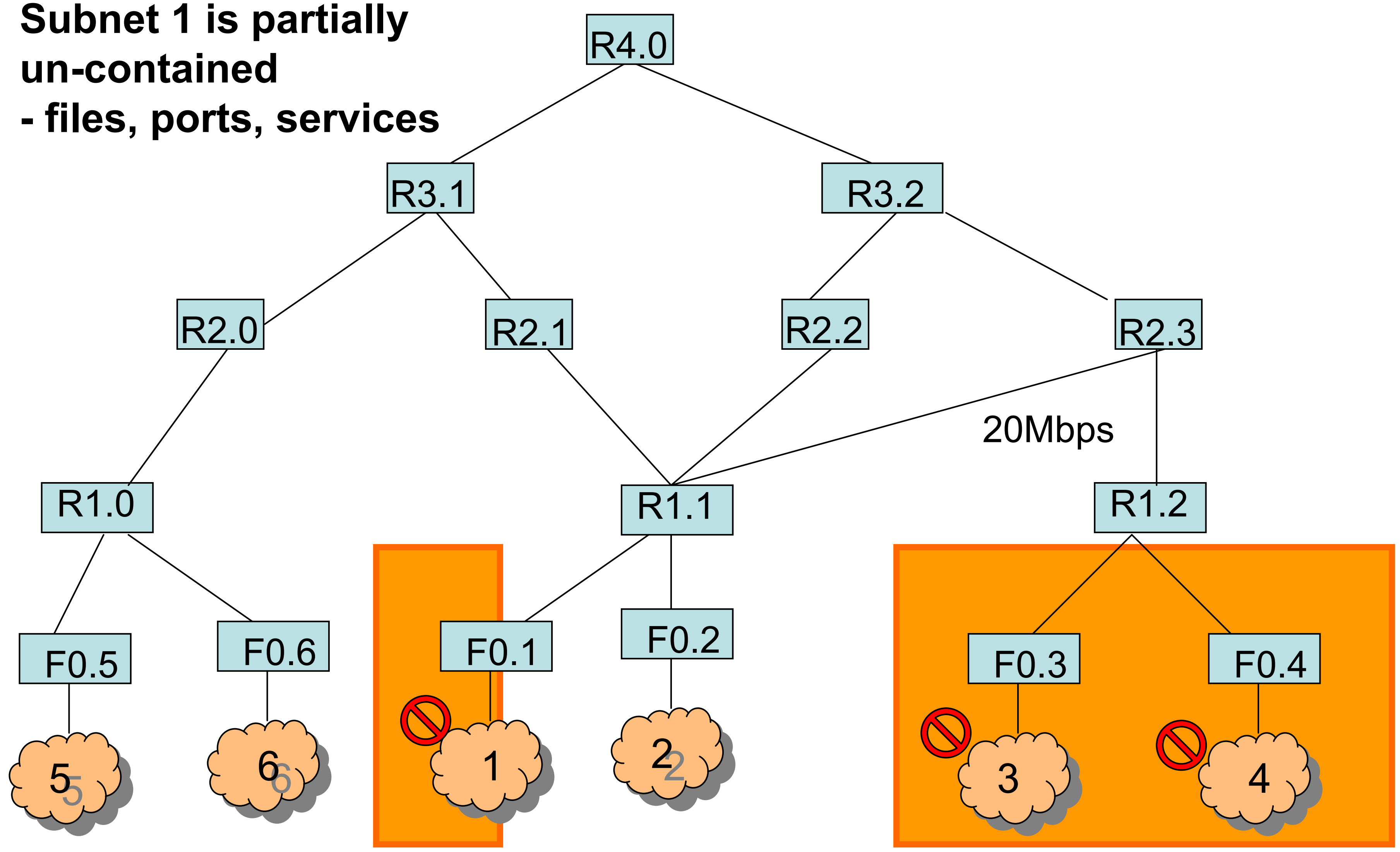
Our approach in a nutshell (3)

- Routers are un-contained
-Updates on file X are isolated



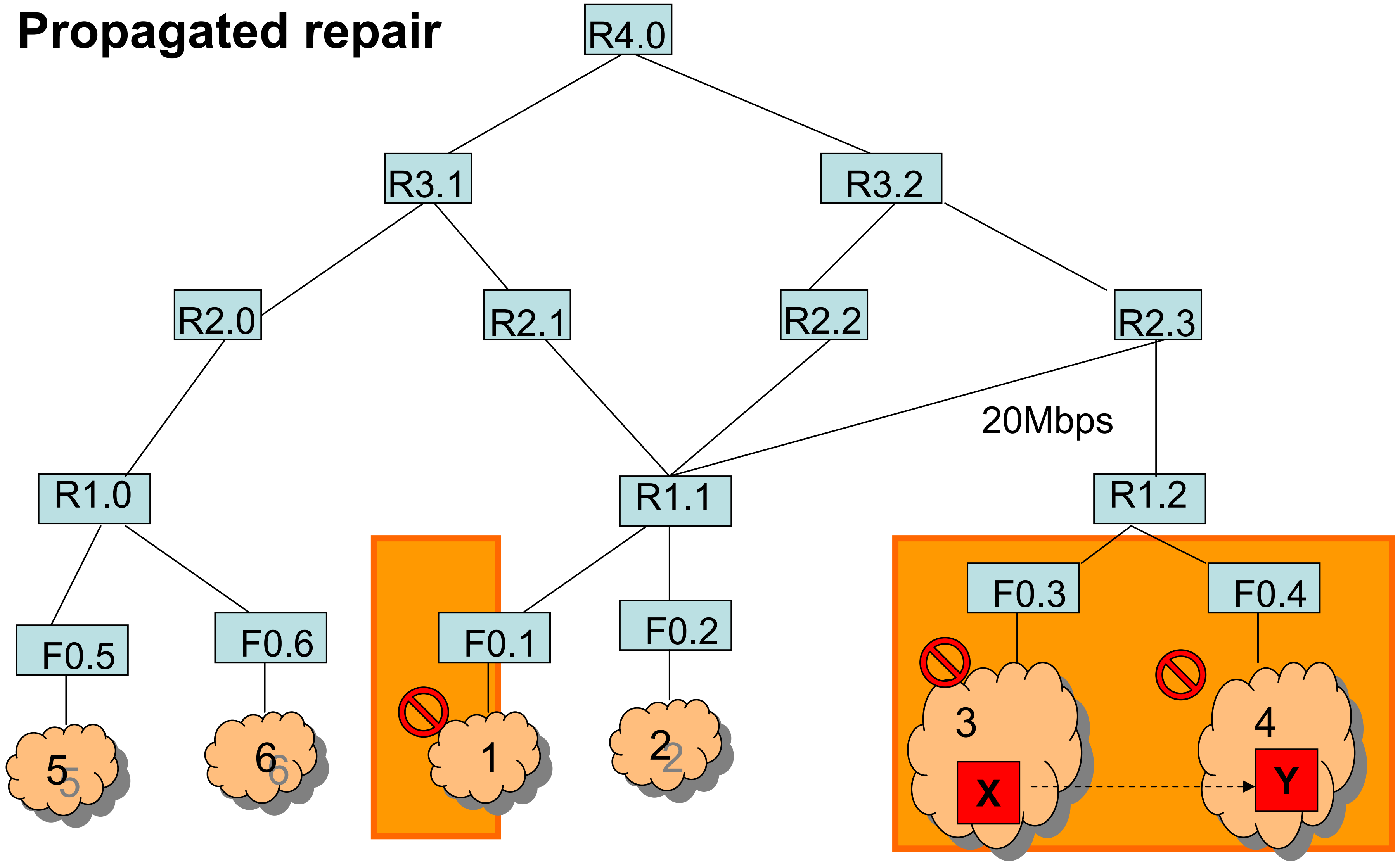
Our approach in a nutshell (4)

Subnet 1 is partially un-contained
- files, ports, services



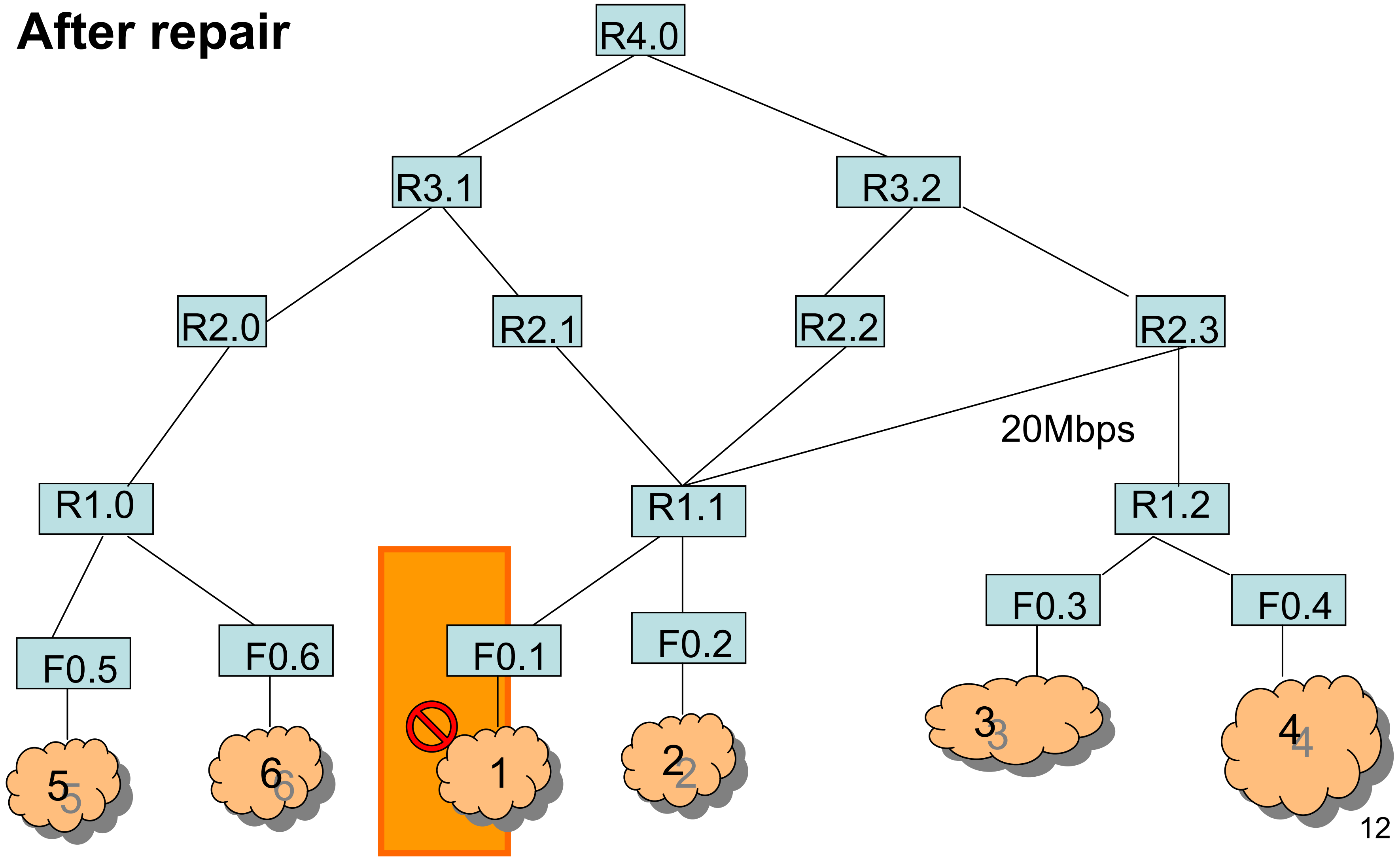
Our approach in a nutshell (5)

Propagated repair



Our approach in a nutshell (6)

After repair



Why our approach can provide substantial availability

Analysis timeline	Self-healing operations
"Something is wrong!"	Time-based and/or distance-based, subnet-level containment; Start isolation; and so on;
… …	… …
Know the port number	Port-level un-containment; port-level packet filtering; add wrappers; and so on
… …	… …
Know which kinds of files are corrupted	Start propagated repair; Adjust isolation operations; and so on

-- No need to wait for results of accurate analysis
-- Can resume services based on rough analysis

Why our approach can minimize the integrity loss

The answer lies in how we do:

- Multiphase containment -- shortly
- Isolation
  - ✓ Minimal integrity loss
  - ✓ When a suspicious thread wants to update or delete a file, the update or delete operation will be transparently isolated in such a way that the original file is still available to trustworthy threads
- Propagated repair
  - ✓ is quick
  - ✓ is concurrent & simultaneous

Multiphase containment

Un-containment with minimal info:

- Time-based un-containment
  - ✓ If a file is not updated since the time the worm happens, then the file will not be corrupted
- Distance-based un-containment
  - ✓ If subnet A is farther from the heart of the worm than subnet B, then B should be affected first probabilistically
- Traffic-based un-containment
  - ✓ If the traffic of my subnet is not increased significantly, my subnet is fine

Note: although in many cases we are not sure whether a worm affects a subnet, in many cases it is clear that a subnet or a host is clean

Key features of our approach

•(1) As soon as a worm alarm is raised, our approach can instantly contain the affected part of the MilEN;

•(2) Our approach enforces multiphase containment: the first phase is very quick, but it can over-contain; the later on phases will make the containment more and more accurate;

•(3) Our approach uses formal dependency analysis to accurately locate the affected part with agility;

•(4) The recovery process is on-the-fly without shutting down many subnets and systems; substantial MilEN services can be sustained;

•(5) Our approach enforces multi-granularity containment: port-level, service-level, protocol-level, file-level, OS-level, DBMS-level, subnet-level, etc;

• (6) Our approach uses propagated recovery to repair propagated worms;

• (7) Our approach does not allow any (physical) deletes in the MilEN so all the info is available during recovery;

• (8) To provide more availability, our approach enforces two novel approaches, namely masking and isolation, when we suspect but are not sure that a worm has been propagating.

Questions?

Thank you!